修士学位論文

題目

抽象構文木とグラフ畳み込みネットワークを用いた類似ソースコード検索

指導教員
井上 克郎 教授

報告者
藤原 裕士

令和 2 年 2 月 5 日

大阪大学 大学院情報科学研究科
コンピュータサイエンス専攻 ソフトウェア工学講座

令和元年度 修士学位論文

抽象構文木とグラフ畳み込みネットワークを用いた類似ソースコード検索

藤原 裕士

内容梗概

近年，大規模なソフトウェアを高品質かつ短期間で開発することが要求されている．このため，開発者は頻繁に既存のソースコードを再利用し，効率的にソフトウェアを開発する．開発者の再利用作業を支援する為に類似ソースコードを検索する様々な手法が提案されており，開発者はこの手法を用いることで再利用対象のソースコードを素早く見つけることができる．しかし，多くの既存手法は自然言語処理手法を用いるため，ソースコードの構造に関する情報が欠落し，類似コード検出漏れが発生する可能性がある．そこで本研究では，抽象構文木とグラフ畳み込みネットワークを用いた類似ソースコード検索手法を提案する．提案手法はグラフ畳み込みネットワークを用いて抽象構文木を学習させることで，ソースコードの構造情報に基づいた，より正確な類似ソースコード検索が期待できる．評価実験では，3つのオープンソースソフトウェアと大規模類似ソースコードベンチマーク BigCloneBench に提案手法を適用し，提案手法が既存手法より高い精度で類似ソースコードを検索できることを確認した．また，提案手法は既存手法では検索困難だった類似ソースコードも検索できることを確認した．

主な用語

ソースコード検索
抽象構文木
グラフ畳み込みネットワーク

目 次

1　まえがき

既存ソフトウェアは，ソフトウェア開発における重要な資源である [1, 2, 3, 4, 5]．近年，大規模なソフトウェアを高品質かつ短期間で開発することが要求されるため，開発者は頻繁に既存ソフトウェアのソースコードを再利用し，効率的にソフトウェアを開発する．また，信頼性の高い既存ソフトウェアのソースコードを再利用することで，新規ソフトウェアの信頼性を担保することができる．そのため，ソフトウェア開発の際に，再利用可能なソースコードを特定することは重要である [6, 7, 8, 9, 10, 11]．しかし，品質の高い再利用可能なソースコードを特定し，再利用することは簡単な作業ではない．例えば，再利用候補のソースコードを Q&A サイト（例：Stack Overflow[1]）等から特定することがあるが，そのソースコードに脆弱性が存在するなどの問題を抱えている可能性がある．この問題を防ぐため，類似ソースコード検索を利用し脆弱性対策が施されている類似ソースコードを見つけることで，ソースコード再利用におけるセキュリティ面の安全性が高まる．また，再利用候補のソースコードの性能が低い場合に類似ソースコード検索を利用し，再利用候補のソースコードに類似した機能を持ち，さらに優れた性能を持つソースコードを見つけて再利用することで，ソフトウェアの性能を向上させることができる．

このように，ソースコード検索はソフトウェア開発において重要な役割を果たす．そのため，ソフトウェア開発の支援を目的とした様々なソースコード検索手法が現在まで提案されている．特に近年では，構文的に類似したソースコード検索手法の他に，実装は異なるが同じ機能を持つソースコード，いわゆる意味的に類似したソースコードの検索を目的とした様々な手法が提案されている [12, 13]．しかし，多くの既存手法では，ソースコード全体に対して自然言語処理（Natural Language Processing，以下 NLP）の手法を適用する．NLP の手法をソースコードに適用する場合，ソースコードの構造に関する情報が欠落するという問題点がある．

そこで本研究では，抽象構文木（Abstract Syntax Tree，以下 AST）とグラフ畳み込みネットワーク（Graph Convolution Networks，以下 GCN）を用いたコード検索手法を提案する．本手法では，ソースコードの AST をグラフとして扱い，グラフを学習するための深層学習モデルである GCN を用いて，ソースコードを機能毎に分類することで，ソースコードの検索を実現する．本手法は，AST の情報を編集したり圧縮したりすることなくそのまま学習に使用するため，既存手法では抽出が困難な，ソースコードの構造情報を学習に利用することができる．また，本手法では教師あり学習を用いるため，“意味的類似”の演繹的な定義（例：テキスト類似度を定め，類似度が閾値を超えた場合は意味的類似）を行う必要がない．そのかわりに，利用者が，直観的な仮定（例：メソッド名が似ている場合は意味的類

[1] https://stackoverflow.com/

似であるという仮定）を置き，具体的に類似ソースコードのクラスタを作成する（例 : 名称が似ているメソッドに同一ラベルを割当）．そして，そのクラスタを深層学習モデルに学習させることで，深層学習モデルは各クラスタ内のソースコードに共通する特徴を抽出し，利用者が求める意味的類似ソースコードを検索することができるようになる．

検索精度の比較実験では，オープンソースソフトウェア（以下 OSS）や，大規模類似ソースコードベンチマークである BigCloneBench[14]（以下 BCB）に対して本手法と既存手法を適用した．その結果，既存手法よりも高い精度で類似ソースコードを検索できることを確認した．また，深層学習モデルの汎化性能を評価するための実験を実施し，本研究における GCN モデルが一定の汎化性能を持つことを確認した．

以降，2 章では本研究の背景について述べる．3 章では本研究で提案する手法について述べる．4 章では既存手法との検索精度比較実験について述べる．5 章では深層学習モデルの汎化性能評価実験について述べる．最後に，6 章でまとめについて述べる．

2　背景

本研究では，意味的類似ソースコードの検索手法を提案する．本研究で扱う類似ソースコードは，一般的にコードクローンと呼ばれる．また，本手法はソースコード分類手法を応用した手法であり，ソースコードの分類を行う為に，深層学習モデルであるグラフ畳み込みネットワークを用いる．また，深層学習モデルにグラフを入力する際には，グラフの要素をベクトルで表現する必要がある．

従って本章では，本研究の背景として，コードクローン，グラフ畳み込みネットワーク，単語のベクトル化手法，ソースコード分類，既存の意味的類似ソースコード検索手法およびその問題点について述べる．

2.1　コードクローン

コードクローンとは，ソースコード中に含まれる，互いに一致または類似したコード片のことであり，一般的に，コードクローンの存在はソフトウェアの保守を困難にすると言われている [15]．コードクローンの主な発生要因は，既存のソースコードのコピーアンドペーストによる再利用であり，他の発生要因としては，定型処理による発生，コード自動生成ツールによる発生，偶然の一致による発生等が挙げられる [16]．

Roy らはコードクローンの定義として，コードクローン間の違いの度合いに基づき以下の 4 つのタイプに分類している [17]．

タイプ 1

空白やタブの有無，コーディングスタイル，コメントの有無などの違いを除き完全に一致するコードクローン．

タイプ 2

タイプ 1 の違いに加えて，変数名などのユーザー定義名，変数の型などが異なるコードクローン．

タイプ 3

タイプ 2 の違いに加えて，文の挿入や削除，変更などが行われているコードクローン．

タイプ 4

類似した処理を実行するが，構文上の実装が異なるコードクローン

本研究では，構文的な類似ソースコードはタイプ 1〜3 のコードクローンを示し，意味的な類似ソースコードはタイプ 4 のコードクローンを示す．

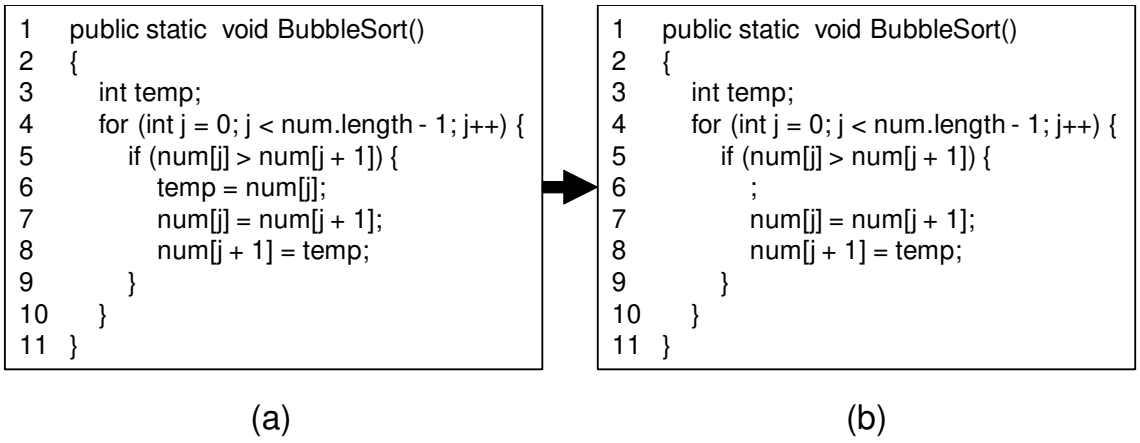

図 1: ミューテーションオペレータ mSDL の例

2.2 類似ソースコード作成を目的としたミューテーション

本研究では，構文的な類似ソースコードを作成する為に，ソースコードに対してミューテーションを適用する．ミューテーションとは，あらかじめ定めておいたルールに基づいてソースコードを変更することであり，一般的にはソフトウェア開発におけるテストケースの評価に用いられている [18]．Roy らはミューテーションを用いて，コードクローン検出ツールを評価する手法を提案している [19]．彼らは，ソースコードの変更ルールをミューテーションオペレータと呼び，以下の 14 種類のミューテーションオペレータを定義している．

mCW 空白の数を変更する．

mCC コメントを変更する．

mCF 改行などのコーディングスタイルを変更する．

mSRI 変数名などのユーザー定義名，変数の型などを規則的に変更する．

mARI 変数名などのユーザー定義名，変数の型などを不規則的に変更する．

mRPE 変数単体の式を別の式に置き換える．

mSIL ある文にわずかな挿入を行う．

mSDL ある文の一部を削除する．

mILs 1 つ以上の文を挿入する．

mDLs 1 つ以上の文を削除する．

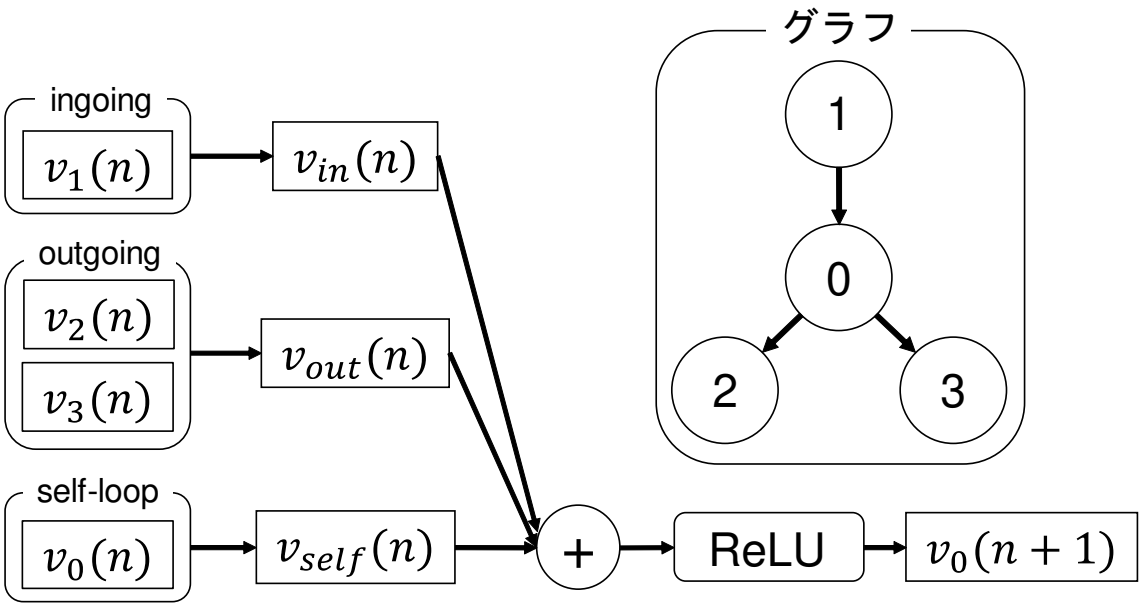

図 2: GCN の畳み込み層の例

mMLs 1 つ以上の文を修正する．

mRDS 宣言文を並べ替える．

mROS 宣言文以外の文を並べ替える．

mCR if 文などの制御構造を別のものに置き換える．

図 1 に，ミューテーションオペレータ mSDL をソースコードに適用した例を示す．図 1（a）の 6 行目の文を削除することで，元のソースコードと構文的に類似したソースコード（図 1（b））が作成された．本研究では，Roy らが定義したミューテーションオペレータをソースコードに適用し，構文的な類似ソースコードを作成する．そして，作成したソースコードは，深層学習モデルに与える学習データとして利用する．

2.3 グラフ畳み込みネットワーク

グラフ畳み込みネットワーク（以下，GCN）[20] とは，グラフの隣接ノードを畳み込んでいくことで，ノードやエッジ，グラフ全体の特徴を抽出するニューラルネットワークである．GCN の畳み込み層の例を図 2 に示す．図 2 右上のグラフ中央のノード 0 の畳み込み $n+1$ 層目におけるベクトル表現 $v_0(n+1)$ を計算する手順について説明する．まず，隣接しているノードの n 層目におけるベクトルと各エッジ（ingoing，outgoing，self-loop）の重みから，中間ベクトル v_{in}，v_{out}，v_{self} を計算する．次に，中間ベクトルを全て足し合わせ，ReLU 等

の活性化関数（ネットワークの出力を補正する関数）に入力する．その結果，$v_0(n+1)$ を得ることができる．このように，GCN を用いるとノード 0 に隣接しているノード 1，2，3 のベクトル表現を加味してノード 0 のベクトル表現を計算することができる．

深層学習モデルには，学習の過程で自動調整される内部パラメータの他に，開発者によって手動で設定されるハイパーパラメータが存在する．ハイパーパラメータは，深層学習モデルの学習の結果や効率に大きく影響する．GCN のハイパーパラメータには，隠れ層のユニット数，ノードベクトルの次元数，畳み込み層の数，学習率，重み減衰，Dropout が存在する．隠れ層のユニット数とは，図 2 における中間ベクトル v_{in}，v_{out}，v_{self} の次元数であり，ネットワークの表現能力や汎化性能に影響する．ノードベクトルの次元数とは，図 2 における v_0，v_1，v_2，v_3 の次元数である．畳み込み層の数とは，図 2 の計算を実行する回数である．学習率は，ネットワークのパラメータを 1 度の調整でどの程度変化させるかを決定する値で，その値が大きすぎるとネットワークのパラメータが収束しなくなり，逆に小さすぎるとパラメータが収束するまでに時間がかかる．重み減衰とは，ネットワークのパラメータの値域にどの程度制限をかけるかを決定する値で，過学習の対策に用いられる．Dropout とは，モデルの学習中にニューラルネットワークのユニットを無効化する確率で，過学習の対策に用いられる．

2.4 単語のベクトル化手法

本節では，単語をベクトル化する手法について説明する．3 章以降で説明する提案手法では，ノードを文書における単語とみなし，以下のベクトル化手法のいずれかを用いて AST のノードをベクトル化し，GCN に入力している．

2.4.1 BoW

Bag of Words（以下，BoW）とは，One-hot ベクトル（ある要素だけが 1，それ以外の要素は 0 であるベクトル）で単語をベクトル化する，伝統的かつ単純なベクトル化手法である．BoW のベクトルの各要素はそれぞれ 1 つの単語に対応し，ある単語をベクトルで表現する場合，その単語に対応する要素だけが 1，他の要素は 0 というベクトルになる．このベクトルの次元は，扱う文書集合全体に出現する単語の種類数に等しい．

2.4.2 LSA

Latent Semantic Analysis（以下，LSA）[21] とは，特異値分解を伴う主成分分析によって文書を解析するための手法である．この手法では，特異値分解を利用し，類似した文脈で使用されることが多い単語をトピックと呼ばれるグループにまとめる．それによって，文書

での使われ方が似ている単語を近い類似度のベクトルにマッピングすることが期待できる．また，特異値分解で求めた特異値が小さい順に特異値行列のトピック列を削除することで，ベクトルの次元数をトピック数の値まで圧縮することが可能である．本研究では，BoW によって作成したベクトルに対して LSA を適用し次元圧縮したベクトルを，LSA によって作成したベクトルと定義する．

2.5 ソースコード分類

本研究では，ソースコード分類を応用することで，ソースコード検索を実現する．そのため，本節では，ソースコード分類の背景について説明する．

ソフトウェアを効率よく開発するために，ソースコード分類は頻繁に用いられている．例えば，ソースコードを自動で機能別に分類することで，大規模なソフトウェアリポジトリに新たに登録されたソースコードに対して機能に関するタグを自動で付与することができる．これによって，開発者にとって必要な機能を持ったソースコードを素早く検索できるようになり，ソフトウェア開発の生産性向上が期待できる．

既存のソースコード分類に関する研究では，記述言語による分類 [22]，コンポーネント間の依存関係による分類 [23]，プログラムの意味（機能性）による分類などが存在する．また，プログラムの意味による分類は様々な粒度で取り組まれており，ソフトウェア単位の分類 [24] や，メソッド単位の分類 [25] などが存在する．これらの分類問題の内，本研究では，メソッド単位での意味的ソースコード分類を扱う．

意味に基づいたメソッド分類の技術は，類似機能を持つメソッドの検索 [12] に用いられる．高精度で意味的メソッドの分類や検索が可能な場合，ソフトウェア開発に必要な機能を持つメソッドの特定や，開発者が使用したいメソッドと同様の機能を持ちながら，信頼性や効率の面でより優れたメソッドの検索など，開発者に対する様々な支援が可能になる．

近年では，ASTNN(a novel AST-based Neural Network)[25] など，深層学習を用いて意味的な類似ソースコード分類に取り組んだ研究が報告されており，これらの研究で利用されている意味的類似ソースコード分類モデルは，基本的に教師あり学習によって実現されている．類似ソースコード分類に教師なし学習を利用する場合,「ソースコードの類似」に関する演繹的な前提を定義する必要がある．ソースコードの構文的な類似は,「ソースコード間で，同じ種類のトークンが同じ順番で連続して並んでいるほど，それらのソースコードは類似度が高い」などで定義することができる．しかし，意味的な類似に関して,「機能が類似している」ことを具体的に言語化し，数式化することは，困難な問題である．このように,「ソースコードの意味的類似」など，演繹的な前提の定義が困難な場合に，教師あり学習を用いることで，教師データに基づき，意味的分類を行うためのルールをモデルが帰納的に学習することができる．

2.6 意味的類似ソースコード検索

2.6.1 既存の意味的類似ソースコード検索手法

近年では，構文的に類似したソースコード検索手法の他に，実装方法は異なるが同じ機能を持つソースコード（意味的に類似したソースコード）検索を目的とした手法が提案されている [12, 13].

FaCoY[12] は，StackOverflow[2]を用いて，意味的類似ソースコードを検索する手法である．具体的には，まず入力として与えられたソースコードに類似したソースコードを StackOverflow から検索する．次に，検索されたソースコードに対応する質問文をクエリとし，再び StackOverflow を用いてソースコードの検索を行う．このような手順で，FaCoY は入力として与えたソースコードと類似したソースコードを検索する．この手法のキーアイディアは，“Q&A サイトにおいて類似した質問文に対応する形で出現するソースコードは意味的に類似している”とすることである．この手法では，入力ソースコードと検索結果のソースコードの間の類似度を算出することで，ランキングを作成している．詳細には，ソースコードに対して TF-IDF を適用することでソースコードをベクトル化し，そのベクトルのコサイン類似度を算出している．

Siamese[13] は，n-gram を用いて類似したソースコードを検索する手法であり，BigClone Bench を用いた評価実験で，構文的な類似度の低いソースコードのペアに対しても，高い精度で検索できることが確認されている．まず，この手法では，ソースコードを以下の 4 種類の異なる配列で表現する．

1. トークンの配列
2. n-gram の配列
3. 識別子・文字列・型を正規化した後の n-gram の配列
4. 括弧・セミコロン以外を正規化した後の n-gram の配列

次に，検索対象ソースコードと入力ソースコードの組において，それぞれの表現方法に対してトークンや N-gram の出現頻度を考慮したスコアを算出する．最後に，閾値よりもスコアが高いソースコードの組を類似ソースコードとして出力する．

2.6.2 既存手法の問題点

既存の意味的類似ソースコード検索手法は，ソースコード全体の特徴を，TF-IDF や n-gram などの NLP の手法を用いて表現している．そのため，ソースコードの構造に関する

[2]https://stackoverflow.com/

情報が欠落する．例えば，TF-IDF ではソースコード中のトークンの出現回数を個別に数えることでソースコード全体をベクトル化するため，トークンの並び順に関する情報は失われる．また，n-gram では，近いトークン同士が同時に出現するという情報は保存されるが，トークンの並び順や，遠いトークン同士の関連情報は失われる．

また，類似ソースコード検索以外のソースコード解析において，ASTNN[25] など，AST と深層学習を利用することで優れた結果を示した研究が存在する．従って，AST に含まれるソースコードの構造情報はソースコードの特徴を表す非常に重要な要素だと考えられる．そこで本研究では，GCN を用いることにより，AST をそのまま学習に使用する．そのため，既存手法では失われていた，ソースコードの構造情報を使用することができる．これにより，既存手法では検索することのできなかった類似ソースコードを検索することが期待できる．

3　提案手法

本研究では，グラフ畳み込みネットワークを用いた類似ソースコード検索手法を提案する．本手法では AST をそのまま学習データとしてモデルに入力する．そのため，ソースコードの構造情報を深層学習に利用することにより，既存手法では検索が困難なソースコードを検索することが期待できる．ソースコードの様々な粒度の中で，メソッドは繰り返し実行する一連の処理を集約することを主な目的として作成され，その再利用の容易さから，再利用の対象になりやすいと考えられる．従って，本研究はメソッド単位のソースコード検索を行う．本提案手法は，多クラス分類モデルを類似ソースコード検索に利用している．この多クラス分類モデルにおけるクラスは互いに類似したメソッドの集合から構成される．同じクラスから抽出した 2 つのメソッドはコードクローンであり，異なるクラスから抽出した 2 つのメソッドはコードクローンではない．各クラスを構成するメソッドを多クラス分類モデルに学習させた後，この学習済みモデルに検索クエリメソッドを入力すると，多クラス分類モデルは入力をいずれかのクラスに分類する．分類先のクラスのメソッドと検索クエリメソッドは類似しているとみなし，分類先クラスのメソッドを検索結果として出力することで，分類モデルによるコード検索を行う．

多クラス分類モデルによって同じクラスに分類されたメソッドの例を図 3 に示す．メソッド (c) は多クラス分類モデルの学習データに含まれており，メソッド (d) とメソッド (e) を多クラス分類モデルを使って分類する場合を考える．メソッド (c) とメソッド (d) は，8 行目以外が一致しており，構文的に類似している．そのため，メソッド (d) は学習済みメソッド (c) が含まれるクラスに分類される．また，学習済みメソッド (c) とメソッド (e) は，構文的に異なるが，どちらも配列をソートするという機能を実現する．そのため，意味的に類似したメソッドであり，メソッド (e) は学習済みメソッド (c) が含まれるクラスに分類される．

本手法では教師あり学習を用いるため，利用手順は STEP T（Training）と STEP C（Classification）の 2 ステップで構成されている．

以降，3.1 節では，用語の定義について述べる．3.2 節では，利用する GCN の仕様について述べる．3.3 節では，分類モデルの学習手順（STEP T）について述べる．最後に，3.4 節では，学習済み分類モデルを用いたソースコード検索手順（STEP C）について述べる．

3.1　用語の定義

類似ソースコードセット　本研究では，構文的または意味的に類似しているメソッドの集合を類似ソースコードセットと定義する．本研究では，類似ソースコードセット中の任意の 2 つのメソッドは，構文的または意味的に類似している．

類似ソースコードセット ID　本研究でソースコード検索に用いるソースコード分類モデル

```
1   public static void bubbleSort(int[] array){
2     int temp;
3     for(int i = 0; i < array.length; i++){
4       for(int j = 0; j< array.length -i -1; j ++){
5         if(array[j] > array[j + 1]){
6           temp = array[j];
7           array[j] = array[j + 1];
8           array[j + 1] = temp;
9         }
10      }
11    }
12  }
```

学習済みメソッド(c)

```
1   public static void bubbleSort(int[] array){
2     int temp;
3     for(int i = 0; i < array.length; i++){
4       for(int j = 0; j< array.length -i -1; j ++){
5         if(array[j] > array[j + 1]){
6           temp = array[j];
7           array[j] = array[j + 1];
8           ;
9         }
10      }
11    }
12  }
```

(d)

```
1   public static void selectionSort(int[] array){
2     for(int i = 0; i < array.length; i++ ){
3       int index = i;
4       for(int j = i + 1; j < array.length; j++){
5         if(array[j] < array[index]){
6           index = j;
7         }
8       }
9       if(i != index){
10        int tmp = array[index];
11        array[index] = array[i];
12        array[i] = tmp;
13      }
14    }
15  }
```

(e)

図 3: 同じクラスに属するメソッドの例

は，学習済みの類似ソースコードセットに検索クエリメソッドを分類する．モデルの学習の時に各類似ソースコードセットに対して固有の数値を割り当て，検索を行うときにモデルにその数値を予測させることで，分類モデルを用いた類似ソースコード検索を行う．ここで，各類似ソースコードセットに対して割り当てられる固有の数値を，本研究では類似ソースコードセット ID と定義する．

学習用データセット　本研究では，モデルの学習に使用するメソッド群を学習用データセットと定義する．このデータセットに含まれるメソッドは，ソースコード検索における検索対象に該当する．

評価用データセット　本研究では，モデルの検証や評価に使用するメソッド群を評価用データセットと定義する．評価実験では，このデータセットに含まれるメソッドを検索クエリとして利用することで，学習用データセットに含まれる類似ソースコードを検索できるかどうかを確認する．

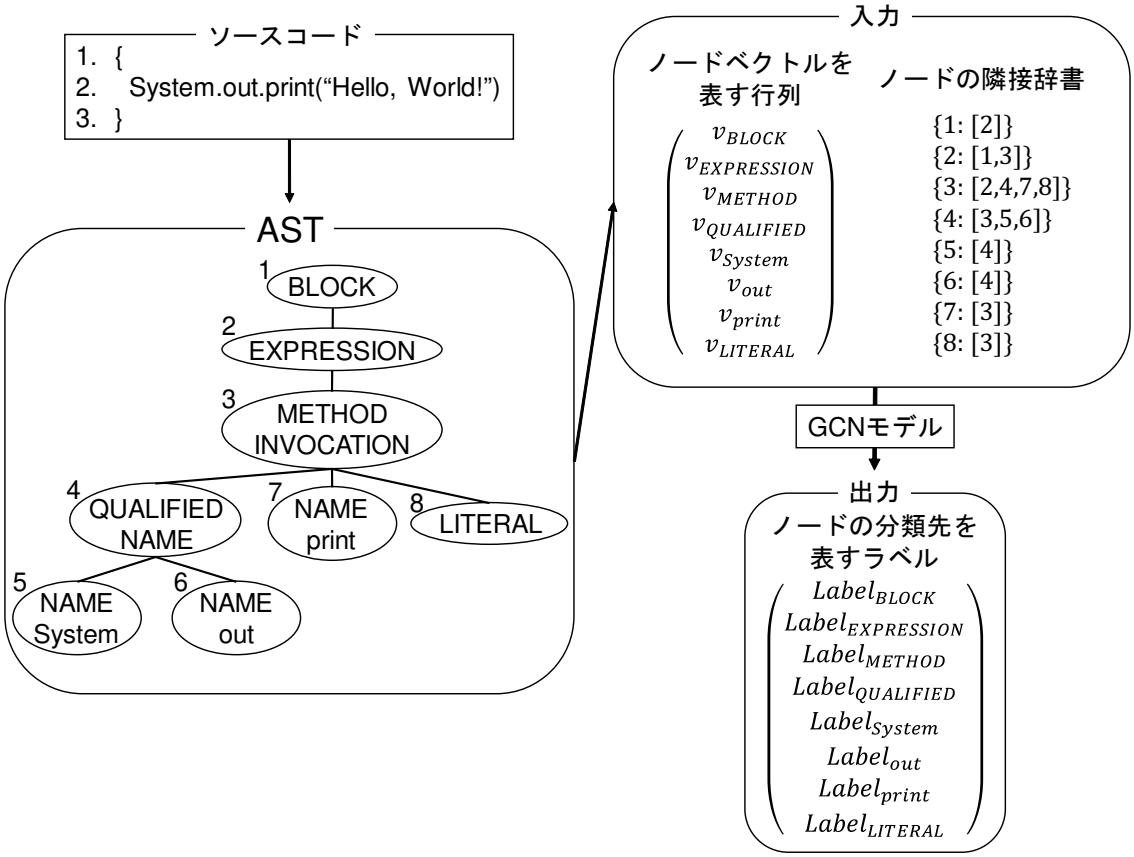

図 4: GCN の入出力概要

3.2 グラフ畳み込みネットワークの仕様

本節では，本手法で利用する GCN の仕様について説明する．本手法における GCN の実装は，Kipf らの実装 [26] を使用する．

3.2.1 入出力の形式

本手法で使用する GCN の入出力の概要を図 4 に示す．本手法ではまず，パーサーを用いてメソッドを AST に変換する．本研究では，Java で記述されているメソッドは EclipseJDT[3] を用いてパースし，C 言語で記述されているメソッドは Antlr[4]を用いてパースした．次に，その AST のノードを深さ優先探索でたどっていき順に番号を割り当て，番号順にノードベクトルを並べて行列を作成する．ここで，AST ノードの個数 a 個，ノードベクトルの次元を b 次元とすると，このノードベクトルを表す行列は $a \times b$ 行列となる．また，番号の割り

[3]https://www.eclipse.org/jdt/
[4]https://www.antlr.org/

当てと同時にノードの隣接辞書を作成する．隣接辞書では，各ノードに対して配列が割り当てられており，各ノードは配列に含まれる番号のノードと隣接している．GCN モデルには，このノードベクトルを表す行列と隣接辞書を入力として与える．

GCN モデルへ入力が与えられると，GCN モデルは，ノードの分類先を表すラベルからなる行列を出力する．つまり，GCN モデルは，メソッド単位の分類ではなく，ノード単位の分類結果を出力する．このノード分類の結果に基づいて，AST の分類結果やランキングを作成する．AST 分類結果の作成方法とランキング作成方法は 3.4 節で説明する．また，分類クラス数を c とすると，この行列は $a \times c$ 行列となる．

3.2.2 ノードベクトルと隣接辞書の作成手順

ノードベクトルを作成するために，2.4 節で説明した単語のベクトル化手法の中で，識別子名を正規化する BoW，識別子名をそのまま利用する BoW，LSA の 3 種類の手法を用いた．LSA は埋め込みベクトルであるのに対して BoW は One-hot ベクトルであるため，モデル内部での計算量は BoW のほうが少なく済むという利点がある．しかし，LSA はベクトルを圧縮し次元数を削減できるのに対し，識別子名をそのまま利用する BoW はベクトルの次元数が非常に多くなるという欠点がある．識別子名を正規化する BoW はベクトルの次元数を少なくできるが，識別子名の文字列情報が欠落するという欠点がある．GCN モデルを利用する際は以上の利点・欠点と計算量の許容範囲やメモリの容量を考慮し，用いるベクトル化手法を 1 つ選択する．

BoW(識別子名正規化) AST ノードを単語とみなし，BoW を用いてノードをベクトル化する．ベクトルの次元数は，AST ノードの種類数に等しい．このとき，識別子名の情報は正規化し，NAME ノードは全て同じノードとして扱う．

BoW(識別子名利用) AST ノードに加え，異なる識別子名を持つ NAME ノードを個別の単語とみなし，BoW を適用してノードをベクトル化する．ベクトルの次元数は，AST ノードの種類数 + 識別子名の種類数となる．

LSA 本手法における LSA の適用方法は特殊である．AST ノードは対応する文字列が無い場合があるため，AST ノード全てを文書の形式に当てはめて LSA でベクトル化することは困難である．従って，本手法では，NAME ノード以外の AST ノードは BoW でベクトル化し，識別子名を持つ NAME ノードは自然言語文書の形式に当てはめて LSA でベクトル化する．そして，ノードベクトルの次元数を揃えるため，パディングを行う．具体的には，BoW で作成したベクトルには，LSA ベクトルと同じ次元数のゼロベクトルを後ろに連結し，LSA で作成したベクトルには，BoW ベクトルと同じ次元数のゼロベクトルを前に連結する．パ

ディングを行うことにより，最終的なベクトルの次元数は，AST ノードの種類数と LSA ベクトルの次元数（トピック数）の合計となる．

また，隣接辞書を作成するためのアルゴリズム Algorithm 1 を示す．Algorithm 1 の詳細な説明は以下の通りである．

隣接辞書作成アルゴリズム Algorithm 1 の入力は AST を表す文字列である．図 4 の AST を文字列として表現すると，(8 (21 (32 (40 (86) (87)) (88) (45)))) になる．この AST 文字列中の数字は，AST ノードの種類に対応する ID を表している．11 行目までの for ループで，各ノードの親ノード番号を登録した配列 parent_array を作成する．親ノードが存在しない場合，親ノード番号の代わりに-1 を登録する．図 4 の AST 文字列を入力した場合，parent_array は [-1,1,2,3,4,4,3,3] となる．12 行目以降で，parent_array の 2 番目以降を順に確認し，親子ノード間の隣接情報を隣接辞書に登録する．この手順によって，図 4 右上に示しているノードの隣接辞書を作成することができる．

Algorithm 1 隣接辞書作成アルゴリズム

Input: AST_string 例：(8 (21 (32 (40 (86) (87)) (88) (45))))

Output: ノードの隣接辞書

```
1: parent_node = -1
2: parent_array = [ ]
3: AST_stream = AST_string.split(' ')
4: for node in AST_stream do
5:     if node[0] ==' (' then
6:         parent_array.append(parent_node)
7:         parent_node = len(parent_array) - 1
8:     else
9:         parent_node = parent_array[parent_node]
10:    end if
11: end for/*parent_array：[-1,1,2,3,4,4,3,3]*/
12: adj_dict = defaultdict(list)
13: for i in range(1, len(parent_array)) do
14:    adj_dict[i].append(parent_array[i])
15:    adj_dict[parent_array[i]].append(i)
16: end for
17: return adj_dict
```

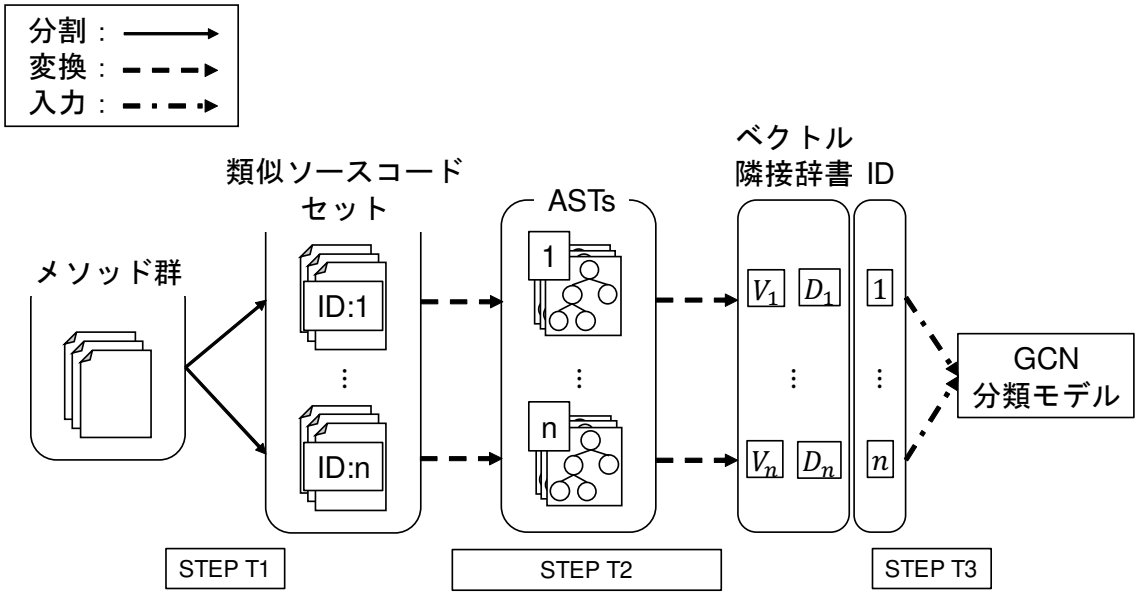

図 5: STEP T の概要

3.3 分類モデルの学習手順

本節では，分類モデルの学習手順 STEP T について説明する．この STEP では，説明変数を AST，目的変数を類似ソースコードセット ID とし，教師あり学習によって GCN の学習を行い，ソースコード分類モデルを作成する．STEP T の概要を図 5 に示す．

STEP T1 学習データとなるメソッドをメソッドの名前に従ってクラスタリングすることで類似ソースコードセットを構築し，各類似ソースコードセットに対して固有の類似ソースコードセット ID を付与する．

STEP T2 各メソッドをノードベクトル行列と隣接辞書の形式に変換する．

STEP T3 ノードベクトル行列と隣接辞書を説明変数，類似ソースコードセット ID を目的変数として，教師あり学習による GCN の学習を行い，ソースコード分類モデルを作成する．

3.4 学習済み分類モデルを用いたソースコード検索手順

本節では，学習済み分類モデルを用いたソースコード検索手順 STEP C について説明する．学習済み分類モデルに検索クエリメソッドの AST を入力すると，そのメソッドに対する類似ソースコードセット ID 推測結果が出力される．この ID が示す類似ソースコードセットを検索結果として出力する．STEP C の概要を図 6 に示す．

STEP C1 検索クエリメソッドをノードベクトル行列と隣接辞書の形式に変換する．

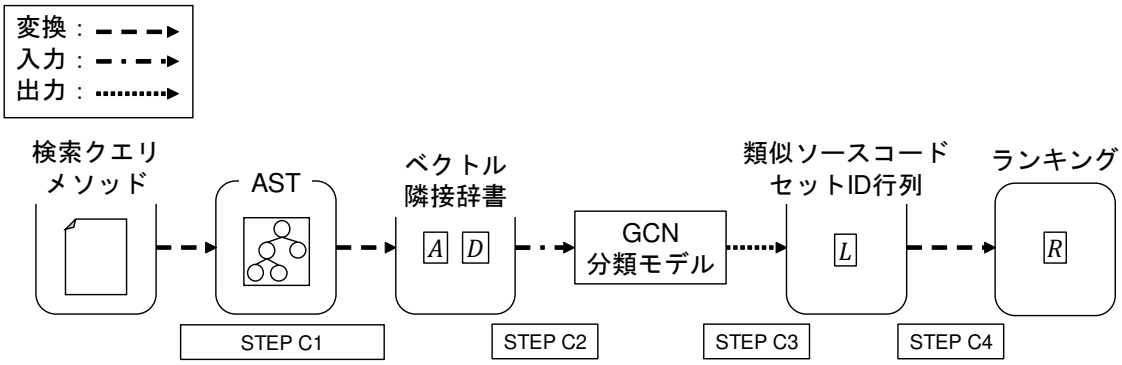

図 6: STEP C の概要

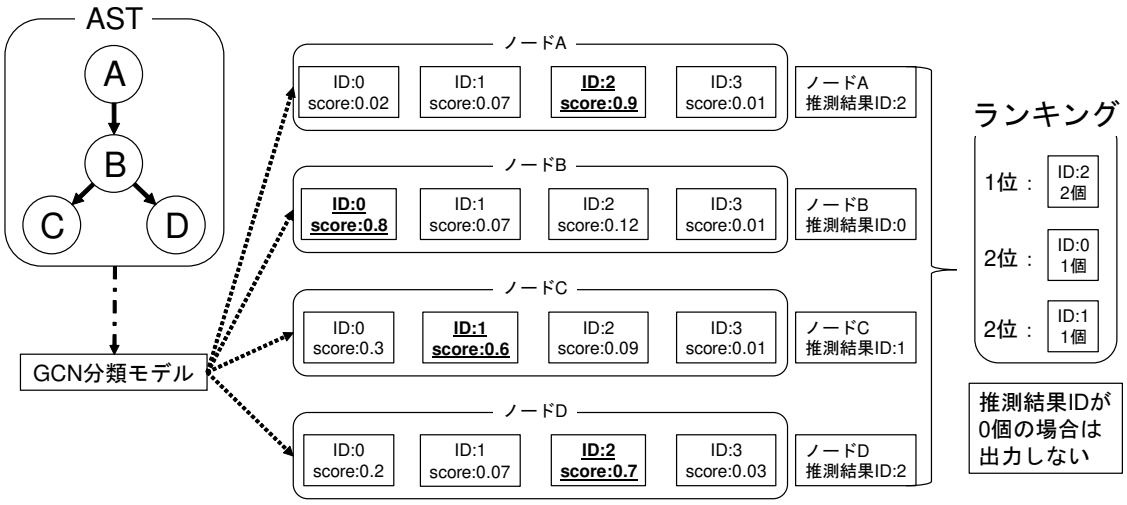

図 7: STEP C4 の概要

STEP C2 ノードベクトル行列と隣接辞書をソースコード分類モデルに入力する．

STEP C3 各ノードの分類先クラスを示す類似ソースコードセット ID 行列が出力される．

STEP C4 類似ソースコードセット ID 行列を解析し，検索結果のランキングを作成する．

STEP C4 の概要を図 7 に示す．STEP C4 では，まず，GCN 分類モデルの出力を参照し，出力スコアが最も高い ID を選択することで，ノードの ID 推測結果を個別に算出する．次に，ノードの推測結果 ID の個数を数え，大きい順に ID を並べることで検索結果のランキングを出力する．この時，推測結果 ID が 0 個の類似ソースコードセットはランキングには含まない．

表 1: 実験環境

OS	Ubuntu 16.04.6 LTS
CPU	Intel(R)Xeon(R) CPU E5-2623 v4 2.60GHz
GPU	NVIDIA Tesla P100 16GB 1.30GHz
ライブラリ	Tensorflow 1.13.1[5]

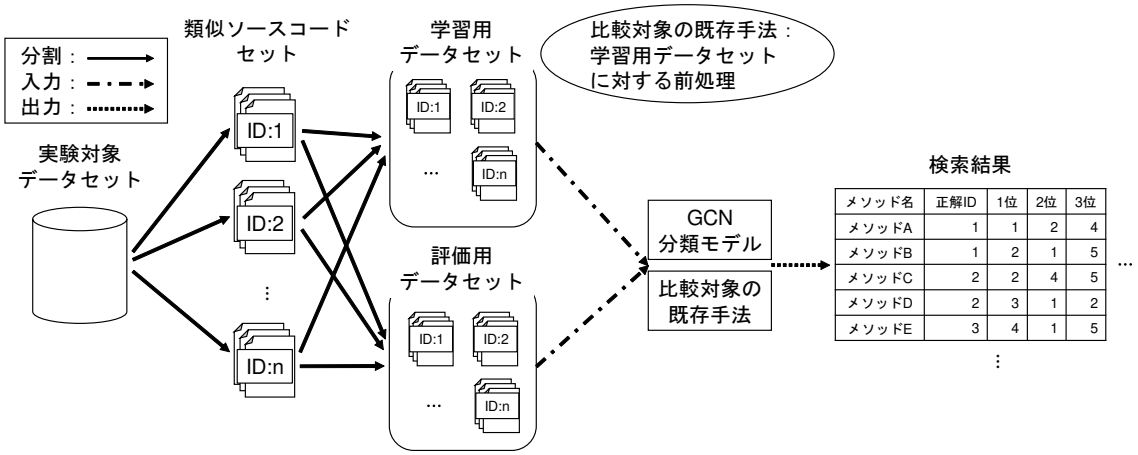

図 8: 検索精度比較実験の概要

4 既存手法との検索精度比較実験

この章では，OSS と BCB を用いて行った既存手法との検索精度評価実験について説明する．本評価実験では評価尺度として Top-k を用いる．Top-k とは，ツールによって出力された検索結果を類似度などのスコア順に並べるなどの方法でランキング形式にしたときに，正解の検索結果が k 位以内に含まれている割合である．また，本評価実験を行った環境を表 1 に示す．

また，検索精度比較実験の概要を図 8 に示す．まず，実験対象リポジトリから類似ソースコードセットを構成し，各類似ソースコードセットを学習用データセットと評価用データセットに分割する．類似ソースコードセットを構成する方法とデータセットの分割方法は 4.1 節で説明する．次に，学習用データセットを用いて GCN 分類モデルを学習する．この時，既存手法は，学習用データセットに含まれるメソッドに対して前処理を行う．最後に，評価用データセットに含まれるメソッドを GCN 分類モデルと既存手法のツールに検索クエリとして入力し，検索結果から Top-k を算出・比較する．

[5]https://www.tensorflow.org/

4.1 評価対象データセット

オープンソースソフトウェア OSS を用いた精度評価では，Apache Maven[6]，Apache Ant[7]，OpenSSL[8]において，バージョン間で編集が行われたメソッドを利用する．OSS を用いた精度評価の目的は，編集後のメソッドを検索クエリとして入力したときに，最も古いバージョンのメソッドを出力できるかを調査することである．まず，同じプロジェクトの各バージョン間において，同じ名称を持つメソッドは，類似機能を持つと考えられる．そこで，バージョン間で編集が行われている同一名称を持つメソッドを集めて，類似ソースコードセットを構成する．本評価ではメモリの都合により，構成した類似ソースコードセットからランダムに 20 個だけ選択し，それぞれに固有の類似ソースコードセット ID を割り当てる．次に，各類似ソースコードセットに含まれる最も古いバージョンのメソッドを学習用データセットと定義する．このとき，学習データ数は各類似ソースコードセットにつき 1 個ずつの合計 20 個で非常に少ないため，それぞれのメソッドに 2.2 節のミューテーションを適用し，作成したメソッド群を学習用データセットに追加する．ここで，作成したメソッドに紐づく類似ソースコードセット ID は，ミューテーションの基になったメソッドの類似ソースコードセット ID と同じとする．また，追加するメソッドの数は，文献 [27] で提案された強化学習に基づいて決定する．次に，20 個選択した類似ソースコードセットに含まれるメソッドを評価用データセットと定義する．つまり，OSS を用いた精度評価では，評価用データセットに含まれる編集後のメソッドを検索クエリとしたときに，学習用データセットに含まれる最も古いバージョンのメソッドを検索できる割合を調べる．

BigCloneBench BCB はメソッド単位のコードクローンからなるベンチマークであるため，BCB を用いた評価実験ではメソッド単位のソースコード検索精度を調べる．BCB には機能クラスタが 43 種類あるため，本評価実験では，そのクラスタを類似ソースコードセットとみなす．各類似ソースコードセットに含まれるメソッドを 8:2 に分割し，8 割を学習用データセット，2 割を評価用データセットに用いる．そして，評価用データセットに含まれるメソッドを検索クエリとしたときに，正しい類似ソースコードセットを検索できる割合を調べる．ただし BCB には，各類似ソースコードセットに含まれるメソッドの数は均等ではないという問題がある．この問題はデータ不均衡問題 [28, 29, 30] であり，この問題を考慮せずに学習用データセットに含まれるすべてのメソッドを学習に利用すると，うまく学習が進まない類似ソースコードセットが出現する．この問題を軽減するため，OSS の場合と同様に，文献 [27] で提案された強化学習に基づいて学習に利用するメソッドの数を決定する．

[6] https://maven.apache.org/
[7] https://ant.apache.org/
[8] https://www.openssl.org/

4.2 比較対象となる既存手法

比較対象となるソースコード検索の既存手法として Siamese と FaCoY を選択する．しかし，これらの手法と提案手法は単純に比較することができない．比較対象となる既存手法は検索クエリメソッドと検索対象メソッドを 1 対 1 対応させるのに対し，提案手法は検索クエリメソッドと検索対象類似ソースコードセットに含まれるメソッドを 1 対多対応させるためである．そのため，比較対象の既存手法では，評価用データセットに含まれるメソッドを検索クエリとしたときに，正しい類似ソースコードセットに含まれるメソッドが一つでも検索結果に出現したら検索成功とする．

4.3 ハイパーパラメータ

評価実験を行うにあたって，OpenSSL のデータセットを用いて GCN モデルのハイパーパラメータを調整した．ハイパーパラメータの探索はグリッドサーチを用いて行った．グリッドサーチとは，ハイパーパラメータ群の候補を規則的に決め，各パラメータの組み合わせを順番に試していく手法である．その結果，GCN のグラフ畳み込み層の数を 4 層，グラフ畳み込みネットワークの隠れ層のユニット数を 128，dropout を 0.2 に設定した場合に最も GCN モデルの予測精度が良くなった．また，他のハイパーパラメータは予測精度にほとんど影響しなかった．そのため，本評価実験では，GCN のグラフ畳み込み層の数を 4 層，グラフ畳み込みネットワークの隠れ層のユニット数を 128，dropout を 0.2 に設定し，その他のハイパーパラメータはデフォルト値を用いて以降の評価実験を行う．また，最適化アルゴリズムは Adam[31] を利用した．

4.4 オープンソースソフトウェアを用いた精度評価の結果

OSS を用いた精度評価結果を表 2 に示す．ただし，FaCoY は対応言語が Java のみであるため，C 言語のプログラムである OpenSSL に適用することはできなかった．Apache Maven では，Top-3，Top-5，All については FaCoY の 0.985 が全手法において最も高い値だったが，Top-1 については提案手法の 0.908 が最も高い値となり，既存手法の精度を上回るという結果になった．また，Apache Ant に対しては，提案手法の Top-1 は 0.905，Top-3，Top-5，All は 1.000 を記録し，既存手法の精度を上回った．OpenSSL に対しては，データセットの難易度が低かった可能性があり，全ての手法で軒並み高い検索精度を記録した．提案手法は Siamese では検索出来なかったメソッドの検索に成功し，その検索精度は 1.000 を記録した．

表 2: オープンソースソフトウェアを用いた精度評価

対象プロジェクト: Apache Maven	Top-1	Top-3	Top-5	All
Siamese	0.693	0.693	0.693	0.693
FaCoY	0.905	**0.985**	**0.985**	**0.985**
提案手法（BoW・識別子名正規化）	0.703	0.800	0.806	0.897
提案手法（BoW・識別子名利用）	**0.908**	0.937	0.943	0.960
提案手法（LSA・50 トピック）	0.874	0.925	0.948	0.948
提案手法（LSA・200 トピック）	0.862	0.925	0.937	0.937
対象プロジェクト: Apache Ant	Top-1	Top-3	Top-5	All
Siamese	0.626	0.626	0.626	0.626
FaCoY	0.854	0.878	0.883	0.883
提案手法（BoW・識別子名正規化）	0.714	0.791	0.883	0.947
提案手法（BoW・識別子名利用）	**0.905**	**1.000**	**1.000**	**1.000**
提案手法（LSA・50 トピック）	0.873	**1.000**	**1.000**	**1.000**
提案手法（LSA・200 トピック）	0.828	0.946	0.973	0.973
対象プロジェクト: OpenSSL	Top-1	Top-3	Top-5	All
Siamese	0.990	0.990	0.990	0.990
FaCoY	×	×	×	×
提案手法（BoW・識別子名正規化）	0.947	0.977	0.977	0.977
提案手法（BoW・識別子名利用）	**1.000**	**1.000**	**1.000**	**1.000**
提案手法（LSA・50 トピック）	**1.000**	**1.000**	**1.000**	**1.000**
提案手法（LSA・200 トピック）	0.982	**1.000**	**1.000**	**1.000**

4.5 大規模コードクローンベンチマーク BigCloneBench を用いた精度評価の結果

学習データとして使用するメソッドの数が多いため，ベクトルの次元数が多くなると，メモリエラーが発生して学習を行うことができなかった．そのため，BCB を用いた評価実験では，識別子名を正規化した BoW と 50 トピックの LSA でノードのベクトル化を行った．BCB を用いた精度評価結果を表 3 に示す．Top-1 については Siamese が，Top-3 に関しては FaCoY が他の手法よりも高い値を記録したが，提案手法は Top-5 で 0.952，All で 0.978 を記録し，既存手法の精度を上回るという結果になった．

識別子名を正規化した BoW をノードベクトルに利用したときの検索精度が低いのは，BCB のコードクローン定義が原因だと考えられる．BCB では，2 つのメソッド間で似たような意味を持つ識別子名が利用されている場合，その 2 つのメソッドのペアはタイプ 4 コード

表 3: BigCloneBench を用いた精度評価

	Top-1	Top-3	Top-5	All
Siamese	**0.848**	0.897	0.908	0.925
FaCoY	0.840	**0.940**	0.950	0.968
提案手法（BoW，識別子名正規化）	0.508	0.648	0.700	0.744
提案手法（LSA，50 トピック）	0.760	0.918	**0.952**	**0.978**

クローンの候補になる．そのため，BCB 内の類似ソースコード，特にタイプ 4 のコードクローンを検索するためには，識別子名に注目することが重要である．このような理由から，識別子名を正規化する手法を BCB に適用した場合，検索精度は低下すると考えられる．

4.6 考察

既存手法では検索できなかったが提案手法だと検索できたメソッドの例を図 9 に示す．この 2 つのメソッドは同じ名前を持っているため，本評価実験では類似ソースコードと定義される．この 2 つのメソッド間で異なる識別子名が多く利用されており，全体的に異なる構造ではあるが，わずかに 3-5 行目が類似している．既存手法はソースコードの構造に着目しておらず，ソースコード全体を参照して類似度を算出するため，図 9 の例のようなメソッド中のごく一部が類似しているケースを見逃してしまうことがあると考えられる．提案手法は，AST に着目することにより，検索対象メソッドと検索クエリメソッドの間に存在する共通の部分木を検知することができるため，図 9 のような，共通部分が少ない場合でも検索を行うことができると考えられる．

次に，提案手法で検索に失敗したメソッドの例を図 10 に示す．メソッド (h) とメソッド (i) は同じ名称を持つため，本評価実験では類似ソースコードと定義される．そのため本評価実験では，メソッド (h) を GCN モデルに入力したときにメソッド (i) が出力されることを期待した．これら 2 つのメソッドは構造が異なり，共通で使用されている識別子名は少ないため，意味的に類似したソースコードと考えられる．これに対して，メソッド (h) とメソッド (j) の間には，類似した構文や識別子が使用されている．特に，buffer.append() の構造はメソッド (j) で非常に多く出現するため，GCN モデルがその構造を学習する回数も多いと推測できる．そのため，GCN モデルは，メソッド (h) に出現する buffer.append() の構造がメソッド (j) の部分木であると推測し，誤答に至ったと考えられる．

以上の例から，本提案手法は，構文木や使用されている識別子名のわずかな共通部分を検知することができるため，メソッドの大部分が変化していても，検索を行うことができるという特徴があると考えられる．また，本提案手法は，意味的類似よりも構文的類似に強く反

```
1.  protected void formatError(String type,Test
2.  test,Throwable error){
3.    if (test != null) {
4.      endTest(test);
5.    }
6.    resultWriter().println(formatTest(test) + ":" + type);
7.    resultWriter().println(error.getMessage());
8.    error.printStackTrace(resultWriter());
9.    resultWriter().println("");
10.}
```

(f) 検索対象

```
1.  private void formatError(String type,Test
2.  test,Throwable t){
3.    if (test != null) {
4.      endTest(test);
5.    }
6.    Element nested=doc.createElement(type);
7.    if (test != null) {
8.      currentTest.appendChild(nested);
9.    }
10. else {
11.     rootElement.appendChild(nested);
12.   }
13.   String message=t.getMessage();
14.   if (message != null && message.length() > 0) {
15.     nested.setAttribute(“message”,
16.       xmlEscape(t.getMessage()));
17.   }
18.   nested.setAttribute("type",
19.     xmlEscape(t.getClass().getName()));
20.   StringWriter swr=new StringWriter();
21.   t.printStackTrace(new PrintWriter(swr,true));
22.   Text trace=doc.createTextNode(swr.toString());
23.   nested.appendChild(trace);
24.}
```

(g) 検索クエリ

図 9: 提案手法でのみ検索できたメソッドの例

応する傾向があると考えられる．意味的な類似ソースコードをさらに検出できるようにする場合，識別子名の英単語としての意味を考慮することは 1 つの方法だと考えられる．図 10 の例であれば，buffer,message,writer,print といった，メッセージ出力に関係する単語が共通して含まれていることを考慮した推測を行うことで，期待通りの検索結果を得ることができる．そのような考慮を行う方法の 1 つとして，Word2Vec[32] など，単語の意味を捉えることを目的とした NLP の技術を，提案手法で用いるベクトル化手法に取り入れることが考えられる．

```
1. private static String createMessage(String
   message,String projectId,File pomFile){
2.   StringBuilder buffer=new StringBuilder(256);
3.   buffer.append(message);
4.   buffer.append(" for project ").append(projectId);
5.   if (pomFile != null) {
6.     buffer.append(" at
   ").append(pomFile.getAbsolutePath());
7.   }
8.   return buffer.toString();
9. }
```

(h) 検索クエリ

```
1. private static String
   createMessage(List<ProjectBuildingResult>
   results){
2.   StringWriter buffer=new StringWriter(1024);
3.   PrintWriter writer=new PrintWriter(buffer);
4.   writer.println("Some problems were encountered
   while processing the POMs:");
5.   for ( ProjectBuildingResult result : results) {
6.     for (   ModelProblem problem :
   result.getProblems()) {
7.       writer.print("[");
8.       writer.print(problem.getSeverity());
9.       writer.print("] ");
10.      writer.print(problem.getMessage());
11.      writer.print(" @ ");
12.      writer.println(problem.getLocation());
13.    }
14. }
15. writer.close();
16. return buffer.toString();
17.}
```

(i) 求める検索結果

```
1. public boolean visitEnter(DependencyNode node){
2.   StringBuilder buffer=new StringBuilder(128);
3.   buffer.append(indent);
4.   org.eclipse.aether.graph.Dependency dep=node.getDependency();
5.   if (dep != null) {
6.     org.eclipse.aether.artifact.Artifact art=dep.getArtifact();
7.     buffer.append(art);
8.     buffer.append(':').append(dep.getScope());
9.     if ((node.getManagedBits() & DependencyNode.MANAGED_SCOPE) ==
         DependencyNode.MANAGED_SCOPE) {
10.      final String premanagedScope=
           DependencyManagerUtils.getPremanagedScope(node);
11.      buffer.append(" (scope managed from ");
12.      buffer.append(StringUtils.defaultString(premanagedScope,"default"));
13.      buffer.append(')');
14.    }
15.    if ((node.getManagedBits() & DependencyNode.MANAGED_VERSION) ==
         DependencyNode.MANAGED_VERSION) {
16.      final String premanagedVersion=
           DependencyManagerUtils.getPremanagedVersion(node);
17.      buffer.append(" (version managed from ");
18.      buffer.append(StringUtils.defaultString(premanagedVersion,"default"));
19.      buffer.append(')');
20.    }
21.    if ((node.getManagedBits() & DependencyNode.MANAGED_OPTIONAL) ==
         DependencyNode.MANAGED_OPTIONAL) {
22.      final Boolean premanagedOptional=
           DependencyManagerUtils.getPremanagedOptional(node);
23.      buffer.append(" (optionality managed from ");
24.      buffer.append(StringUtils.defaultString(premanagedOptional,"default"));
25.      buffer.append(')');
26.    }
27.    if ((node.getManagedBits() & DependencyNode.MANAGED_EXCLUSIONS) ==
         DependencyNode.MANAGED_EXCLUSIONS) {
28.      buffer.append(" (exclusions managed)");
29.    }
30.    if ((node.getManagedBits() & DependencyNode.MANAGED_PROPERTIES) ==
         DependencyNode.MANAGED_PROPERTIES) {
31.      buffer.append(" (properties managed)");
32.    }
33.  }
34. else {
35.    buffer.append(project.getGroupId());
36.    buffer.append(':').append(project.getArtifactId());
37.    buffer.append(':').append(project.getPackaging());
38.    buffer.append(':').append(project.getVersion());
39.  }
40.  logger.debug(buffer.toString());
41.  indent+="   ";
42.  return true;
43.}
```

(j) 実際の検索結果

図 10: 提案手法で目的の検索結果を得ることができなかった例

5　深層学習モデルの汎化性能評価実験

4 章の評価実験にて，本提案手法の一定の有効性を示すことができた．しかし，4 章の評価実験では，同一プロジェクト内のソースコードから学習用データセットと評価用データセットを作成している．そのため，学習データと評価データの間に依存関係が存在しており，GCN モデルが過学習を起こしているという懸念がある．そのため，この章では，依存関係をできるだけ排除した学習・評価用データセットを作成し，GCN モデルの汎化性能を評価する．

5.1　データセット

学習用データセット　この評価実験では，4.1 章で準備した OpenSSL の学習用データセットを利用する．

評価用データセット　依存関係をできるだけなくし，かつ評価における正解を定義するため，まず，OpenSSL から派生したソフトウェアである，BoringSSL[9]と LibreSSL[10]のソースコードから，学習に用いた OpenSSL のメソッドと同名のメソッドを抽出し，評価用データセットに追加する．この結果，27 個のメソッドが評価用データセットに追加された．

次に，OpenSSL と完全に依存関係がないソフトウェアの 1 つである Apache httpd[11]から，OpenSSL のメソッドと同名または類似した意味を持つメソッドを評価用データセットに追加する．この基準に従い，Apache httpd からは．OpenSSL のメソッド `args_verify`・`chopup_args` に類似した意味の名称を持つメソッド `check_args`・`split_argv` を抽出し，評価用データセットに追加した．この結果，9 個のメソッド `check_args` と，6 個のメソッド `split_argv` が評価用データセットに追加された．

5.2　依存関係を排除した評価用データセットを用いた精度評価

この汎化性能評価実験の概要を図 11 に示す．本実験は以下の手順で行う．

1. 学習用データセットを用いて GCN モデルの学習を行う．今回は，4 章の評価実験中に作成した OpenSSL 用のソースコード検索モデルを利用する．
2. 評価用データセットに含まれるメソッドを検索クエリとして学習済み GCN モデルに入力し，対応するメソッドを学習用データセット内から検索する精度を Top-k を用いて評価する．

[9]https://boringssl.googlesource.com/
[10]https://www.libressl.org/
[11]https://httpd.apache.org/

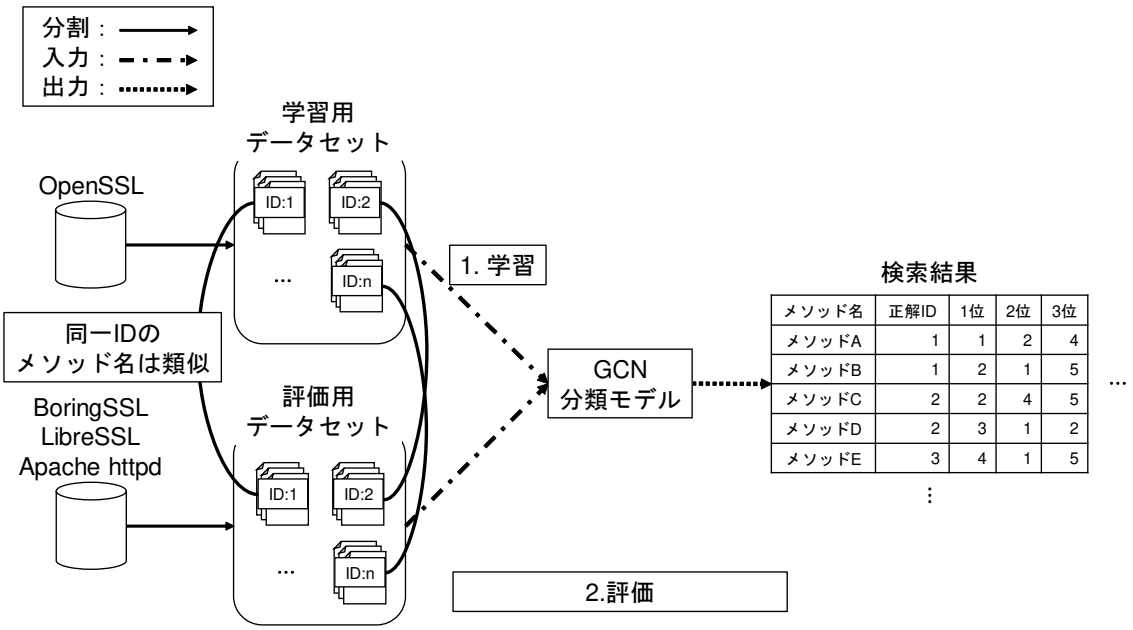

図 11: 汎化性能評価実験の概要

表 4: 依存関係を排除した評価用データセットを用いた精度評価の結果

	Top-1	Top-3	Top-5	All
提案手法（BoW・識別子名正規化）	0.790	**0.930**	**0.977**	**1.000**
提案手法（BoW・識別子名利用）	**0.791**	0.907	**0.977**	**1.000**
提案手法（LSA・50 次元）	0.769	0.769	0.923	**1.000**
提案手法（LSA・200 次元）	0.718	0.744	0.974	0.974

精度評価結果を表 4 に示す．4 章の評価実験の結果と比べて検索精度は低下しているが，減少幅は小さい．このように，依存関係のないデータセットに対しても高い精度で検索を行うことができた．また，4 章の実験結果と異なり，識別子名を正規化した場合と識別子名を利用した場合で検索精度にあまり差が生じなかった．

検索に成功したメソッドの例を図 12 に示す．メソッド (k) は学習データとして使用したメソッドであり，OpenSSL に含まれている．メソッド (l) は OpenSSL と全く無関係のプロジェクト Apache httpd から抽出したメソッドである．この 2 つのメソッドの名称は類似した意味を持っている．メソッド (l) を検索クエリとして GCN モデルに入力したところ，メソッド (k) を検索結果として出力した．このように，提案手法では，類似した処理を行っていると考えられるメソッドを検索することができる．

5.3 考察

本節では，汎化性能の評価実験結果に基づいて，過学習についての考察を行う．本研究における過学習とは，学習データに対して深層学習モデルが適合しすぎてしまうために，学習データと依存関係のない，GCN モデルにとって未知のデータを入力したときに，予測精度が低下してしまい，汎化性能が不足している状態を指す [33]．汎化性能の評価実験では，学習データとして用いたメソッドと同じ機能を持つが，ソフトウェアの派生によって構文的な差異が生まれたメソッド，すなわち，学習データと依存関係のないメソッドを検索クエリとして GCN モデルに入力した．また，学習データと非常に名称の意味が似ているが，全く無関係のプロジェクトから収集したメソッドを検索クエリとして GCN モデルに入力した．その結果，4 章の評価実験の結果と比べて検索精度は低下したものの，その減少幅は小さかった．この点から，分類器モデルの能力に悪影響を及ぼすほどの重度の過学習は発生していないと思われる．よって，提案手法の GCN モデルには汎化性能がある．つまり，学習済みのデータだけに適合しているわけではなく，AST の特徴を捉え，同じ特徴を持つメソッドを検出する能力を持つと考えられる．

また，4 章の実験結果と異なり，識別子名を正規化した場合と識別子名を利用した場合で検索精度にあまり差が生じなかった．同一プロジェクト内のメソッドと比べて，異なるプロジェクト間のメソッドにおいて同一識別子の出現数は減少すると考えられる．そのため，識別子名情報が十分に存在せず，結果的には正規化した場合と似たような状況になったと考えられる．

```
1.  int args_verify(char ***pargs, int *pargc,
2.        int *badarg, BIO *err, X509_VERIFY_PARAM **pm){
3.    ASN1_OBJECT *otmp = NULL;
4.    unsigned long flags = 0;
5.    int i;
6.    int purpose = 0;
7.    char **oldargs = *pargs;
8.    char *arg = **pargs, *argn = (*pargs)[1];
9.    if (!strcmp(arg, "-policy")){
10.     if (!argn)  *badarg = 1;
11.     else{
12.       otmp = OBJ_txt2obj(argn, 0);
13.       if (!otmp){
14.         BIO_printf(err, "Invalid Policy ¥"%s¥"¥n",
15.                    argn);
16.         *badarg = 1;
17.       }
18.     }
19.     (*pargs)++;
20.   }
21.   else if (strcmp(arg,"-purpose") == 0){
22.     X509_PURPOSE *xptmp;
23.     if (!argn)  *badarg = 1;
24.     else{
25.       i = X509_PURPOSE_get_by_sname(argn);
26.       if(i < 0){
27.         BIO_printf(err, "unrecognized purpose¥n");
28.         *badarg = 1;
29.       }
30.       else{
31.         xptmp = X509_PURPOSE_get0(i);
32.         purpose = X509_PURPOSE_get_id(xptmp);
33.       }
34.     }
35.     (*pargs)++;
36.   }
37.   else if (!strcmp(arg, "-ignore_critical"))
38.     flags |= X509_V_FLAG_IGNORE_CRITICAL;
39.   else if (!strcmp(arg, "-issuer_checks"))
40.     flags |= X509_V_FLAG_CB_ISSUER_CHECK;
41.   else if (!strcmp(arg, "-crl_check"))
42.     flags |=  X509_V_FLAG_CRL_CHECK;
43.   else if (!strcmp(arg, "-crl_check_all"))
44.     flags |= X509_V_FLAG_CRL_CHECK|X509_V_FLAG_CRL_CHECK_ALL;
45.   else if (!strcmp(arg, "-policy_check"))
46.     flags |= X509_V_FLAG_POLICY_CHECK;
47.   else if (!strcmp(arg, "-explicit_policy"))
48.     flags |= X509_V_FLAG_EXPLICIT_POLICY;
49.   else if (!strcmp(arg, "-x509_strict"))
50.     flags |= X509_V_FLAG_X509_STRICT;
51.   else if (!strcmp(arg, "-policy_print"))
52.     flags |= X509_V_FLAG_NOTIFY_POLICY;
53.   else
54.     return 0;
55.   if (*badarg){
56.     if (*pm)  X509_VERIFY_PARAM_free(*pm);
57.     *pm = NULL;
58.     goto end;
59.   }
60.   if (!*pm && !(*pm = X509_VERIFY_PARAM_new())){
61.     *badarg = 1;
62.     goto end;
63.   }
64.   if (otmp)  X509_VERIFY_PARAM_add0_policy(*pm, otmp);
65.   if (flags)  X509_VERIFY_PARAM_set_flags(*pm, flags);
66.   if (purpose)  X509_VERIFY_PARAM_set_purpose(*pm, purpose);
67.   end:
68.   (*pargs)++;
69.   if (pargc)  *pargc -= *pargs - oldargs;
70.   return 1;
71. }
```

(k)OpenSSL に含まれるメソッド

```
1.  static void check_args(apr_pool_t *pool, int argc, const char *const argv[],
2.            int *alg, int *mask, char **user, char **pwfilename,
3.            char **password){
4.    const char *arg;
5.    int args_left = 2;
6.    int i;
7.    if (argc < 3) {
8.      usage();
9.    }
10.   for (i = 1; i < argc; i++) {
11.     arg = argv[i];
12.     if (*arg != '-') {
13.       break;
14.     }
15.     while (*++arg != '¥0') {
16.       if (*arg == 'c') {
17.         *mask |= NEWFILE;
18.       }
19.       else if (*arg == 'n') {
20.         *mask |= NOFILE;
21.         args_left--;
22.       }
23.       else if (*arg == 'm') {
24.         *alg = ALG_APMD5;
25.       }
26.       else if (*arg == 's') {
27.         *alg = ALG_APSHA;
28.       }
29.       else if (*arg == 'p') {
30.         *alg = ALG_PLAIN;
31.       }
32.       else if (*arg == 'd') {
33.         *alg = ALG_CRYPT;
34.       }
35.       else if (*arg == 'b') {
36.         *mask |= NONINTERACTIVE;
37.         args_left++;
38.       }
39.       else {
40.         usage();
41.       }
42.     }
43.   }
44.   if (*mask & (NEWFILE & NOFILE)) {
45.     apr_file_printf(errfile, "%s: -c and -n options conflict¥n", argv[0]);
46.     exit(ERR_SYNTAX);
47.   }
48.   if ((argc - i) != args_left) {
49.     usage();
50.   }
51.   if (*mask & NOFILE) {
52.     i--;
53.   }
54.   else {
55.     if (strlen(argv[i]) > (PATH_MAX - 1)) {
56.       apr_file_printf(errfile, "%s: filename too long¥n", argv[0]);
57.       exit(ERR_OVERFLOW);
58.     }
59.     *pwfilename = apr_pstrdup(pool, argv[i]);
60.     if (strlen(argv[i + 1]) > (MAX_STRING_LEN - 1)) {
61.       apr_file_printf(errfile, "%s: username too long (>%" APR_SIZE_T_FMT ")¥n",
62.         argv[0], MAX_STRING_LEN - 1);
63.       exit(ERR_OVERFLOW);
64.     }
65.   }
66.   *user = apr_pstrdup(pool, argv[i + 1]);
67.   if ((arg = strchr(*user, ':')) != NULL) {
68.     apr_file_printf(errfile, "%s: username contains illegal character '%c'¥n",
69.         argv[0], *arg);
70.     exit(ERR_BADUSER);
71.   }
72.   if (*mask & NONINTERACTIVE) {
73.     if (strlen(argv[i + 2]) > (MAX_STRING_LEN - 1)) {
74.       apr_file_printf(errfile, "%s: password too long (>%" APR_SIZE_T_FMT ")¥n",
75.         argv[0], MAX_STRING_LEN);
76.       exit(ERR_OVERFLOW);
77.     }
78.     *password = apr_pstrdup(pool, argv[i + 2]);
79.   }
80. }
```

(l)Apache httpd に含まれるメソッド

図 12: 類似した名称をもつメソッドの例

6　まとめ

本研究では，抽象構文木とグラフ畳み込みネットワークを用いて，類似ソースコードを検索する手法を提案した．本提案手法では，抽象構文木をノードベクトル行列と隣接辞書に変換し，類似ソースコードセット ID を用いて教師あり学習を行うことで，類似ソースコードを検索することができるグラフ畳み込みネットワークモデルを作成する．意味的類似ソースコードの検索を目的とした既存手法では，ソースコード全体を自然言語処理の手法を使ってベクトル化するため，ソースコードの構造に関する情報が欠落するという問題点があった．しかし，本手法は，抽象構文木をグラフとしてそのまま深層学習モデルに入力することができ，ソースコードの構造に関する情報を使用することが可能になる．そして，ソースコードの構造情報の分だけ情報量が増えるため，既存手法では検索することのできなかった類似ソースコードを検索することができるようになる．

検索精度比較実験では，3 つのオープンソースソフトウェアと大規模コードクローンベンチマーク BigCloneBench を使用し，検索精度評価指標の 1 つである Top-k の値を既存の類似ソースコード検索手法と比較した．その結果，本提案手法は，Top-1 では 3 つのオープンソースソフトウェアにおいて既存手法よりも高い値を記録した．また，All の値は，Apache Ant，OpenSSL，BigCloneBench において既存手法よりも高数値を記録した．また，実験結果の詳細を調査したところ，既存手法では検索できないが提案手法だと検索できるメソッドの存在を確認することができた．さらに，汎化性能評価実験において学習・評価用データセット間の依存関係を取り除いた実験を行い，GCN モデルの汎化性能を確認した．

謝辞

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上 克郎 教授には，研究の各段階において，多くのご指導を賜りました．井上 教授から多く賜った適切な御指導により，本論文を完成させることができました．井上 教授に心より深く感謝いたします．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松下 誠 准教授には，大変多くの御助言を賜りました．松下 准教授に心より深く感謝いたします．

名古屋大学 大学院情報学研究科附属組込みシステム研究センター 吉田 則裕 准教授には，研究に関する直接の御指導を賜りました．常に適切な御指導及び御助言を頂いたことにより，本論文を完成させることができました．吉田 准教授に心より深く感謝いたします．

京都工芸繊維大学 情報工学・人間科学系 テニュアトラック 崔 恩瀞 助教には，研究や本論文の構成に関する多くの御助言を頂きました．崔 恩瀞 助教に心より深く感謝いたします．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 神田 哲也 助教には，実験環境の構築に関する多くのご助言を頂きました．神田 助教に心より深く感謝いたします．

最後に，私を支えてくださった，大阪大学大学院情報科学研究科コンピュータサイエンス専攻井上研究室の皆様に，心より深く感謝いたします．

参考文献

[1] Andrew J Ko, Brad A Myers, Michael J Coblenz, and Htet Htet Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Transactions on software engineering*, No. 12, pp. 971–987, 2006.

[2] Tao Xie and Jian Pei. Mapo: Mining api usages from open source repositories. In *Proceedings of the 2006 international workshop on Mining software repositories*, pp. 54–57. ACM, 2006.

[3] Rabe Abdalkareem, Emad Shihab, and Juergen Rilling. On code reuse from stackoverflow: An exploratory study on android apps. *Information and Software Technology*, Vol. 88, pp. 148–158, 2017.

[4] Naohiro Kawamitsu, Takashi Ishio, Tetsuya Kanda, Raula Gaikovina Kula, Coen De Roover, and Katsuro Inoue. Identifying source code reuse across repositories using lcs-based source code similarity. In *Proc. of SCAM 2014*, pp. 305–314, 2014.

[5] Takashi Ishio, Raula Gaikovina Kula, Tetsuya Kanda, Daniel M. Germán, and Katsuro Inoue. Software ingredients: detection of third-party component reuse in java software release. In *Proc. of MSR 2016*, pp. 339–350, 2016.

[6] Sushil Krishna Bajracharya and Cristina Videira Lopes. Analyzing and mining a code search engine usage log. *Empirical Software Engineering*, Vol. 17, No. 4-5, pp. 424–466, 2012.

[7] Caitlin Sadowski, Kathryn T Stolee, and Sebastian Elbaum. How developers search for code: a case study. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pp. 191–201. ACM, 2015.

[8] Katsuro Inoue, Yusuke Sasaki, Pei Xia, and Yuki Manabe. Where does this code come from and where does it go?-integrated code history tracker for open source systems. In *Proc. of ICSE 2012*, pp. 331–341, 2012.

[9] Darren Rush and Ankur Bulsara. Source code search engine, December 27 2007. US Patent App. 11/663,417.

[10] Katsuro Inoue, Reishi Yokomori, Tetsuo Yamamoto, Makoto Matsushita, and Shinji Kusumoto. Ranking significance of software components based on use relations. *IEEE Trans. Software Eng.*, Vol. 31, No. 3, pp. 213–225, 2005.

[11] Takashi Ishio, Yusuke Sakaguchi, Kaoru Ito, and Katsuro Inoue. Source file set search for clone-and-own reuse analysis. In *Proc. of MSR 2017*, pp. 257–268, 2017.

[12] Kisub Kim, Dongsun Kim, Tegawendé F. Bissyandé, Eunjong Choi, Li Li, Jacques Klein, and Yves Le Traon. FaCoY: A code-to-code search engine. In *Proc. of ICSE 2018*, pp. 946–957, 2018.

[13] Chaiyong Ragkhitwetsagul and Jens Krinke. Siamese: scalable and incremental code clone search via multiple code representations. *Empirical Software Engineering*, pp. 2236–2284, 2019.

[14] Jeffrey Svajlenko, Judith F Islam, Iman Keivanloo, Chanchal K Roy, and Mohammad Mamun Mia. Towards a big data curated benchmark of inter-project code clones. In *Proc. of ICSME 2014*, pp. 476–480, 2014.

[15] 肥後芳樹, 楠本真二, 井上克郎. コードクローン検出とその関連技術. 電子情報通信学会論文誌, Vol. J91-D, No. 6, pp. 1465–1481, 2008.

[16] Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant Anna, and Lorraine Bier. Clone detection using abstract syntax trees. In *Proceedings of International Conference on Software Maintenance*, pp. 368–377, 1998.

[17] Chanchal K. Roy, James R. Cordy, and Rainer Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, Vol. 74, No. 7, pp. 470–495, 2009.

[18] Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *IEEE transactions on software engineering*, Vol. 37, No. 5, pp. 649–678, 2010.

[19] Chanchal K Roy and James R Cordy. A mutation/injection-based automatic framework for evaluating code clone detection tools. In *Proc. of ICSTW 2009*, pp. 157–166, 2009.

[20] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *Proc. of ESWC 2018*, pp. 593–607, 2018.

[21] Thomas K Landauer and Susan T Dumais. A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review*, Vol. 104, No. 2, p. 211, 1997.

[22] Ganesan Kavita and Foti Romano. C# or java? typescript or javascript? machine learning based classification of programming languages. `https://github.co/2Jif7Sg`, 2019.

[23] Reishi Yokomori, Norihiro Yoshida, Masami Noro, and Katsuro Inoue. Use-relationship based classification for software components. In *Proc. of QuASoQ 2018*, pp. 59–66, 2018.

[24] Shinji Kawaguchi, Pankaj K Garg, Makoto Matsushita, and Katsuro Inoue. Mudablue: An automatic categorization system for open source repositories. *Journal of Systems and Software*, Vol. 79, No. 7, pp. 939–953, 2006.

[25] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Kaixuan Wang, and Xudong Liu. A novel neural source code representation based on abstrace syntax tree. In *Proc. of ICSE 2019*, pp. 783–794, 2019.

[26] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proc. of ICLR 2017*, 2017.

[27] 藤原裕士, 崔恩瀞, 吉田則裕, 井上克郎. 深層学習を用いたソースコード分類のための学習用データセット改善手法の提案. ソフトウェアエンジニアリングシンポジウム 2019 ポスター発表.

[28] Chao Chen and Mei-Ling Shyu. Clustering-based binary-class classification for imbalanced data sets. In *Proc. of IRI 2011*, pp. 384–389, 2011.

[29] Yilin Yan, Min Chen, Mei-Ling Shyu, and Shu-Ching Chen. Deep learning for imbalanced multimedia data classification. In *Proc. of ISM 2015*, pp. 483–488, 2015.

[30] Nitesh V. Chawla, Nathalie Japkowicz, and Aleksander Kotcz. Editorial: Special issue on learning from imbalanced data sets. *SIGKDD Explor. Newsl.*, Vol. 6, No. 1, pp. 1–6, June 2004.

[31] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. of ICLR 2015*, 2015.

[32] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proc. of ICLR 2013*, 2013.

[33] Rich Caruana, Steve Lawrence, and Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. *Advances in neural information processing systems*, pp. 402–408, 2001.